Information

INF2440 Vår 2018 eksamen

Dato og tid: 11. juni 2018 09:00.

Varighet: 4 timer

Hjelpemidler: Alt skriftlig materiale er tillatt. Ingen elektroniske hjelpemidler er tillatt. Powerpoint slides er vedlagt som PDF.

1 Question 1.1: Java Thread Creation

I et Java-program må programmøren lage og starte hver tråd som programmet bruker. Er dette korrekt?
Velg ett alternativ

- Nei, den første tråden blir opprettet automatisk og startes automatisk og settes til å utføre main(). De gjenværende trådene må opprettes og startes av en av trådene i programmet.
- Nei, alle tråder startes automatisk.
- Nei, programmereren må bare gi koden for trådene. Kompilatoren genererer automatisk tråder på kompileringstid og sikrer at de startes når programmet kjører.
- Ja, programmet må opprette og starte alle tråder.
- Ja, programmet må opprette og starte alle tråder i main().

Maks poeng: 4

2 Question 1.2: Java Threads Features

Hvilke av de følgende setningene gjelder for Java-tråder som er opprettet i det samme program? (Mer enn ett svar er mulig.)
Velg ett eller flere alternativer

- Java tråder utføres med samme hastighet fordi hver tråd er tilordnet sin egen kjerne.
- Der er aldrig mere enn en Java tråd, som kjører.
- Hvis det er flere tråder enn tilgjengelige kjerner, kan programmet ikke kjøres.
- Java-tråder kan få tilgang til de lokale variabeler i alle tråde.
- Java tråder kan utføres parallelt uavhengig av hverandre på multi-core maskiner.
- Java tråder ligger i en felles del av minnet og kan få tilgang til statiske variabler som er defineret i klassen som inneholder trådene.

Maks poeng: 6

3 Question 1.3: Java Synchronized

Kan vi sørge for at bare én tråd om gangen utfører en metode ved å annotere metoden med **synchronized**.

Velg ett alternativ

- Ja, med unntak av den opprinnelige tråden som kjører main() - og som alltid har lov til å utføre metoden selv om en annen tråd kjører metoden på samme tid.
- Ja, hvis mer enn en tråd kaller metoden, sikrer Java-systemet at bare én tråd utfører metoden om gangen.
- Nei, inne i metoden må vi legge til synkronisering, for eksempel, CyclicBarrier.
- Bare hvis returmetoden for metoden er void.

Maks poeng: 5

4 # Question 1.4: Performance of Java Threads

Hvilke av de følgende uttalelsene er nærmest virkeligheten?

Velg ett alternativ

- Java tråder er dyre å lage, men det blir billigere og billigere å lage tråder, jo flere tråder et program oppretter.
- Java-tråder er billige å lage så lenge det er mindre tråder enn kjerner på maskinen.
- Java tråder tar mye tid å lage og synkronisere, slik at det generelt ikke bør opprettes en tråd, med mindre det kan gjøre mye arbeid, for eksempel minst tusenvis av heltalsadditioner.
- Java-tråder kan opprettes og kjøre så fort at selv om vi bare gjør to eller tre heltalladditioner, er det verdt å gjøre dem i en egen tråd fordi tråden vil utføre raskt på en annen kjerne.

Maks poeng: 5

i # Question 2: Cyclic Barriers

I dette spørsmålet bør du se på programmet Problem.java vedlagt som et PDF-dokument.

5 # Question 2.1: Cyclic Barrier - termination with two threads

Når programmet kalles med parametrene "2 0", vil programmet alltid avslutte? Forklar svaret ditt.

Skriv ditt svar her...

Format

Words: 0

Maks poeng: 10

6 Question 2.2: Cyclic Barrier - output with two threads

Når programmet kalles med parameterne "2 0", vil programmet alltid skrive ut samme tekst? Forklar svaret ditt.

Skriv ditt svar her...

Format

Words: 0

Maks poeng: 10

7 Question 2.3 Cyclic Barrier: Termination with four threads

Når programmet kalles med parameterne "2 2", vil programmet alltid avslutte? Forklar svaret ditt.

Skriv ditt svar her...

Format | B I U x_2 x^2 | I_x | | | | | | | Ω | | | ∑ | ABC |

Words: 0

Maks poeng: 12

8 Question 2.4: Cyclic Barrier: Output with four threads

Når programmet kalles med parameterne "2 2", vil programmet alltid skrive ut samme tekst? Hvis ja, vis teksten. Hvis ikke, gi to forskjellige mulige eksempler på hva programmet skriver ut.

Skriv ditt svar her...

Format | B I U x_2 x^2 | I_x | | | | | | | Ω | | | ∑ | ABC |

Words: 0

Maks poeng: 12

Question 3: Finding Prime Deserts

Dette spørsmålet er basert på Erathostenes Sieve som finner primtal til et gitt nummer N. Du burde allerede være ganske kjent med sikten som du har implementert den i Oblig 3 (vedlagt som PDF).

Dette spørsmålet handler om å finne såkalte Primtals-ørkener, Prime Deserts. En Prime Desert er et intervall [A, B] hvor A < B, A og B er primtall, og det er ingen primtall mellom A og B. Eksempler på slike ørkener er [2, 3], [7, 11], [23, 29] og [337, 347]. Størrelsen på en Prime Desert er definert som B-A-1, dvs. antall heltall strengt mellom A og B. A kalles startpunktet til ørkenen og B kalles sluttpunktet.

Du skal skrive et Java-program som genererer en liste over Primtalls-ørkener. Listen bør sorteres slik at intervallen kommer i stigende rekkefølge, dvs. startpunktene er i stigende rekkefølge. Den første Prime Desert er [2, 3] og deretter skal den neste Prime Desert være større enn den forrige i hele listen. Videre bør du finne alle Prime Deserts der sluttpunktet ikke er større enn N, meni listen bør du utelukke noen Prime Desert, hvis størrelse ikke er større enn størrelsen på den forrige Prime Desert på listen. For eksempel betyr det at starten på listen er: [2,3], [3, 5], [7, 11], [23, 29], [89, 97], ... dvs. [5 , 7], [11, 13], [13, 17], [17, 19] og flere, er utelatt fordi deres størrelse ikke er større enn den forrige Prime Desert på listen. For å være sikker kan du ikke uttlate en Prime Desert, hvis den er større enn den forrige i listen og mindre enn den neste; for eksempel kan du ikke ha en liste som starter med [2, 3] etterfulgt av [23, 29].

Du trenger ikke skrive hele programmet, da du kan anta at du allerede har koden du skrev til Oblig 3. Du er velkommen til å basere koden på Modell2 fra forelesningene i uke 5.

9 # Question 3.1: Sequential Program for finding Prime Deserts

Skriv et sekventielt Java-program som finner listen over Prime Deserts spesifisert i introduksjonen til Spørsmål 3.
Skriv ditt svar her...

```
1
```

Maks poeng: 20

10 # Question 3.2: Parallel Program to find Prime Deserts

Skriv et parallelt Java-program som finner listen over Prime Deserts spesifisert i introduksjonen til Spørsmål 3. Forsøk å oppnå speedup over sekvensiell versjon.

Skriv ditt svar her...

```
1
```

Maks poeng: 40

Question 4: Bubblesort

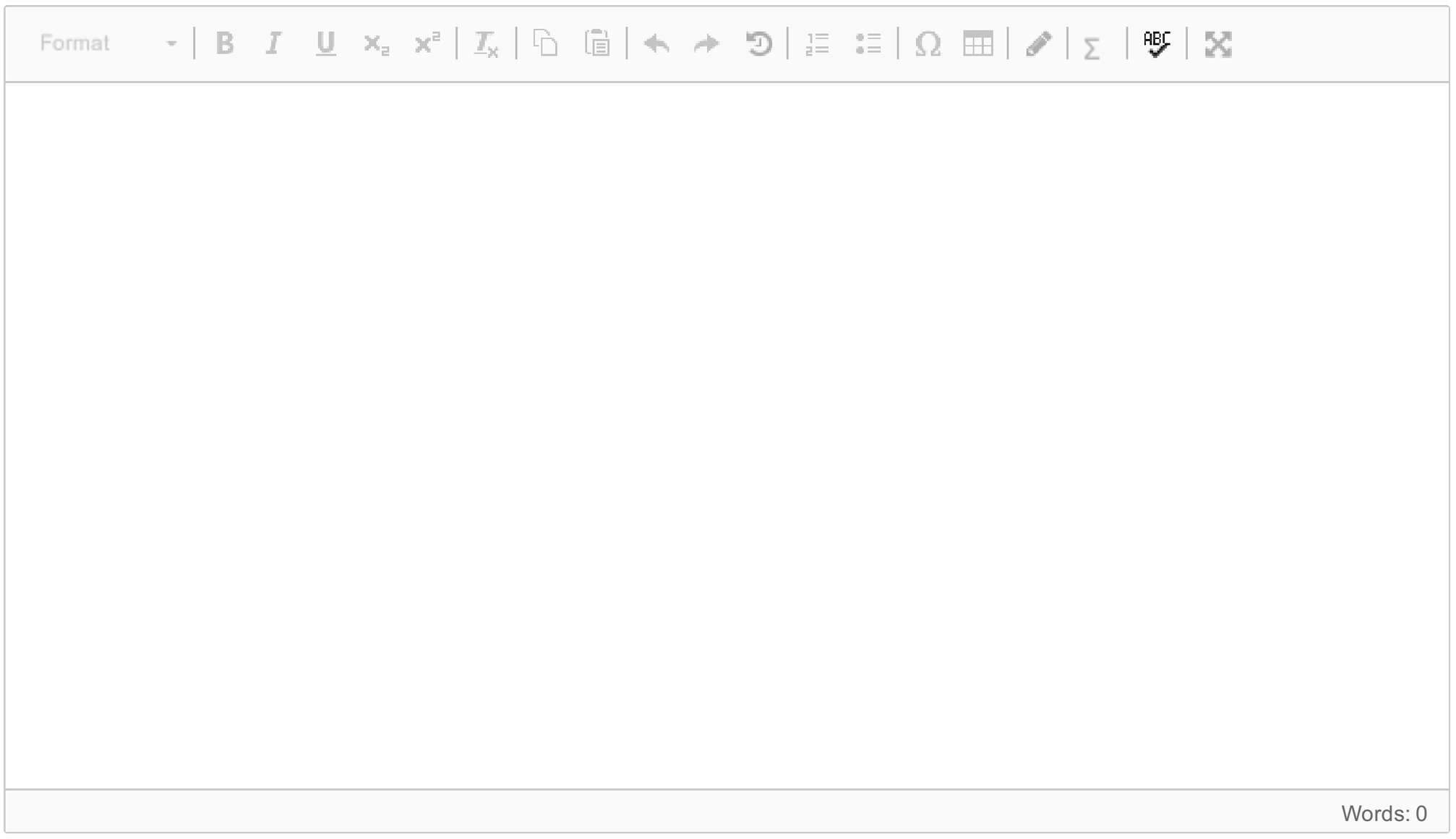
Bubblesort er en sorteringsalgoritme som sorterer for eksempel et heltall-array A, ved å sammenligne to naboelementer og ombytte dem hvis den første er større enn den andre. Sorteringen består av et antall passerer. Hvert pass går gjennom arrayet ett element om gangen og utfører ombytning, om nødvendig. På slutten av pass k er de k største tal på plass øverst i arrayet, så neste pass trenger ikke å vurdere topp k elementene i arrayet.

En sekvensiell bubblesort av et heltall er gitt som en PDF-fil.

11 Question 4.1 Parallelizing Bubblesort

Hvordan kan Bubblesort parallelliseres? Beskriv et design av en løsning.

Skriv ditt svar her...

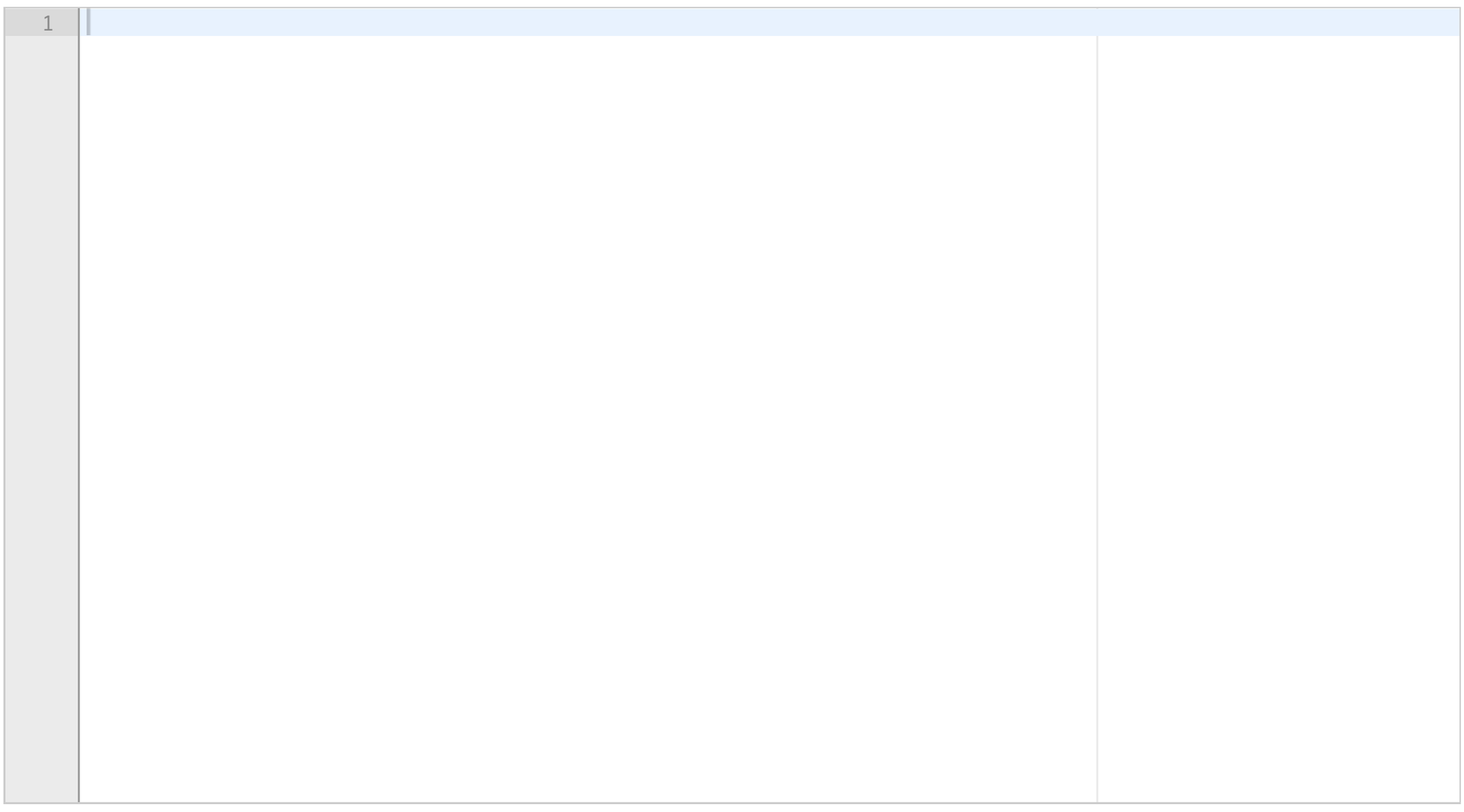


Maks poeng: 20

12 Question 4.2: Parallel Bubblesort

Skriv et Java-program som implementerer designet av en parallell versjon av Bubblesort fra Spørsmål 4.1. Forsøk å få speedup i forhold til den sekvensielle versjon. Legg til en ekstra forklaring som du har som kommentarer i koden.

Skriv ditt svar her...



Maks poeng: 40

13 Question 5.1: Recursion and Parallelism

Det er mange måter å parallellisere en rekursiv algoritme. Av de følgende uttalelsene, hvilken av dem er en god

ide (du kan velge mer enn én):

Velg ett eller flere alternativer

- Det er viktig å balansere antall tråder og mengden arbeid som utføres av hver tråd.
- Når rekursive kall danner en trestruktur, er det en god ide å lage og bruke tråder bare nederst på treet.
- Rekursive kall er størrelsesordener billigere enn å skape og kjøre en tråd.
- En enkel måte å parallellisere et rekursivt program er å ha en tråd per rekursiv kall. Dette fører nesten alltid til å oppnå en god speedup.
- Når rekursive kall danner en trestruktur, er det en god idé å lage ny tråde øverst på treet og begrense antall tråder på bunnen av treet.
- Det er viktig å begrense antall tråder.
- Rekursive kall er relativt dyre i forhold til å skape og kjøre en tråd.
- Når rekursive kall danner en trestruktur, er det en god ide å erstatte grener på de nedre nivåene av treet med enkle sekvensielle løsninger på problemet.

Maks poeng: 16

Question 12
Attached

```
static void bubbleSort(int[] arr) {
   int n = arr.length;
   int temp;
   for (int i=0; i < n; i++){
      for (int j=1; j < (n-i); j++){
          if(arr[j-1] > arr[j]){
             //swap elements
             temp = arr[j-1];
             arr[j-1] = arr[j];
             arr[j] = temp;
          }

      }
   }
}
```

Question 12
Attached

```
static void bubbleSort(int[] arr) {
   int n = arr.length;
   int temp;
   for (int i=0; i < n; i++){
      for (int j=1; j < (n-i); j++){
          if(arr[j-1] > arr[j]){
             //swap elements
             temp = arr[j-1];
             arr[j-1] = arr[j];
             arr[j] = temp;
          }

      }
   }
}
```

Question 5
Attached

```
import java.util.concurrent.*;
class Problem {
   // felles data og metoder A
   static int num = 2;
   static int extra = 2;
   static CyclicBarrier b;

   public static void main(String [] args) {
      Problem p = new Problem();
      num = Integer.parseInt(args[0]);
      extra = Integer.parseInt(args[1]);
      b = new CyclicBarrier(num);

      p.utfoer(num+extra); // extra threads
      System.out.println(" Main TERMINATED");
   } // end main

   void utfoer (int antT) {
      Thread [] t = new Thread [antT];
      for (int i =0; i< antT; i++)
         ( t[i] = new Thread(new Arbeider(i))).start();
     try {
        for (int i =0; i< antT; i++) t[i].join();
     } catch(Exception e) {}
   } // end utfoer

   class Arbeider implements Runnable {
      // lokale data og metoder B
      int ind;

      void sync() {
         try{
            b.await();
         } catch (Exception e) { return;}
      }

      public Arbeider (int in) {ind = in;};

      public void run() {
         sync();
         System.out.println("A"+ind);
         sync();
         System.out.println("B"+ind);
      } // end run
   } // end indre klasse Arbeider
} // end class Problem
```

Question 5
Attached

```
import java.util.concurrent.*;
class Problem {
   // felles data og metoder A
   static int num = 2;
   static int extra = 2;
   static CyclicBarrier b;

   public static void main(String [] args) {
      Problem p = new Problem();
      num = Integer.parseInt(args[0]);
      extra = Integer.parseInt(args[1]);
      b = new CyclicBarrier(num);

      p.utfoer(num+extra); // extra threads
      System.out.println(" Main TERMINATED");
   } // end main

   void utfoer (int antT) {
      Thread [] t = new Thread [antT];
      for (int i =0; i< antT; i++)
         ( t[i] = new Thread(new Arbeider(i))).start();
     try {
        for (int i =0; i< antT; i++) t[i].join();
     } catch(Exception e) {}
   } // end utfoer

   class Arbeider implements Runnable {
      // lokale data og metoder B
      int ind;

      void sync() {
         try{
            b.await();
         } catch (Exception e) { return;}
      }

      public Arbeider (int in) {ind = in;};

      public void run() {
         sync();
         System.out.println("A"+ind);
         sync();
         System.out.println("B"+ind);
      } // end run
   } // end indre klasse Arbeider
} // end class Problem
```

Question 6
Attached

```
import java.util.concurrent.*;
class Problem {
   // felles data og metoder A
   static int num = 2;
   static int extra = 2;
   static CyclicBarrier b;

   public static void main(String [] args) {
      Problem p = new Problem();
      num = Integer.parseInt(args[0]);
      extra = Integer.parseInt(args[1]);
      b = new CyclicBarrier(num);

      p.utfoer(num+extra); // extra threads
      System.out.println(" Main TERMINATED");
   } // end main

   void utfoer (int antT) {
      Thread [] t = new Thread [antT];
      for (int i =0; i< antT; i++)
         ( t[i] = new Thread(new Arbeider(i))).start();
     try {
        for (int i =0; i< antT; i++) t[i].join();
     } catch(Exception e) {}
   } // end utfoer

   class Arbeider implements Runnable {
      // lokale data og metoder B
      int ind;

      void sync() {
         try{
            b.await();
         } catch (Exception e) { return;}
      }

      public Arbeider (int in) {ind = in;};

      public void run() {
         sync();
         System.out.println("A"+ind);
         sync();
         System.out.println("B"+ind);
      } // end run
   } // end indre klasse Arbeider
} // end class Problem
```

Question 6
Attached

```
import java.util.concurrent.*;
class Problem {
   // felles data og metoder A
   static int num = 2;
   static int extra = 2;
   static CyclicBarrier b;

   public static void main(String [] args) {
      Problem p = new Problem();
      num = Integer.parseInt(args[0]);
      extra = Integer.parseInt(args[1]);
      b = new CyclicBarrier(num);

      p.utfoer(num+extra); // extra threads
      System.out.println(" Main TERMINATED");
   } // end main

   void utfoer (int antT) {
      Thread [] t = new Thread [antT];
      for (int i =0; i< antT; i++)
         ( t[i] = new Thread(new Arbeider(i))).start();
     try {
        for (int i =0; i< antT; i++) t[i].join();
     } catch(Exception e) {}
   } // end utfoer

   class Arbeider implements Runnable {
      // lokale data og metoder B
      int ind;

      void sync() {
         try{
            b.await();
         } catch (Exception e) { return;}
      }

      public Arbeider (int in) {ind = in;};

      public void run() {
         sync();
         System.out.println("A"+ind);
         sync();
         System.out.println("B"+ind);
      } // end run
   } // end indre klasse Arbeider
} // end class Problem
```

Question 7
Attached

```
import java.util.concurrent.*;
class Problem {
   // felles data og metoder A
   static int num = 2;
   static int extra = 2;
   static CyclicBarrier b;

   public static void main(String [] args) {
      Problem p = new Problem();
      num = Integer.parseInt(args[0]);
      extra = Integer.parseInt(args[1]);
      b = new CyclicBarrier(num);

      p.utfoer(num+extra); // extra threads
      System.out.println(" Main TERMINATED");
   } // end main

   void utfoer (int antT) {
      Thread [] t = new Thread [antT];
      for (int i =0; i< antT; i++)
         ( t[i] = new Thread(new Arbeider(i))).start();
     try {
        for (int i =0; i< antT; i++) t[i].join();
     } catch(Exception e) {}
   } // end utfoer

   class Arbeider implements Runnable {
      // lokale data og metoder B
      int ind;

      void sync() {
         try{
            b.await();
         } catch (Exception e) { return;}
      }

      public Arbeider (int in) {ind = in;};

      public void run() {
         sync();
         System.out.println("A"+ind);
         sync();
         System.out.println("B"+ind);
      } // end run
   } // end indre klasse Arbeider
} // end class Problem
```

Question 7
Attached

```
import java.util.concurrent.*;
class Problem {
   // felles data og metoder A
   static int num = 2;
   static int extra = 2;
   static CyclicBarrier b;

   public static void main(String [] args) {
      Problem p = new Problem();
      num = Integer.parseInt(args[0]);
      extra = Integer.parseInt(args[1]);
      b = new CyclicBarrier(num);

      p.utfoer(num+extra); // extra threads
      System.out.println(" Main TERMINATED");
   } // end main

   void utfoer (int antT) {
      Thread [] t = new Thread [antT];
      for (int i =0; i< antT; i++)
         ( t[i] = new Thread(new Arbeider(i))).start();
     try {
        for (int i =0; i< antT; i++) t[i].join();
     } catch(Exception e) {}
   } // end utfoer

   class Arbeider implements Runnable {
      // lokale data og metoder B
      int ind;

      void sync() {
         try{
            b.await();
         } catch (Exception e) { return;}
      }

      public Arbeider (int in) {ind = in;};

      public void run() {
         sync();
         System.out.println("A"+ind);
         sync();
         System.out.println("B"+ind);
      } // end run
   } // end indre klasse Arbeider
} // end class Problem
```

Question 8
Attached

```
import java.util.concurrent.*;
class Problem {
   // felles data og metoder A
   static int num = 2;
   static int extra = 2;
   static CyclicBarrier b;

   public static void main(String [] args) {
      Problem p = new Problem();
      num = Integer.parseInt(args[0]);
      extra = Integer.parseInt(args[1]);
      b = new CyclicBarrier(num);

      p.utfoer(num+extra); // extra threads
      System.out.println(" Main TERMINATED");
   } // end main

   void utfoer (int antT) {
      Thread [] t = new Thread [antT];
      for (int i =0; i< antT; i++)
         ( t[i] = new Thread(new Arbeider(i))).start();
     try {
        for (int i =0; i< antT; i++) t[i].join();
     } catch(Exception e) {}
   } // end utfoer

   class Arbeider implements Runnable {
      // lokale data og metoder B
      int ind;

      void sync() {
         try{
            b.await();
         } catch (Exception e) { return;}
      }

      public Arbeider (int in) {ind = in;};

      public void run() {
         sync();
         System.out.println("A"+ind);
         sync();
         System.out.println("B"+ind);
      } // end run
   } // end indre klasse Arbeider
} // end class Problem
```

Question 8
Attached

```
import java.util.concurrent.*;
class Problem {
   // felles data og metoder A
   static int num = 2;
   static int extra = 2;
   static CyclicBarrier b;

   public static void main(String [] args) {
      Problem p = new Problem();
      num = Integer.parseInt(args[0]);
      extra = Integer.parseInt(args[1]);
      b = new CyclicBarrier(num);

      p.utfoer(num+extra); // extra threads
      System.out.println(" Main TERMINATED");
   } // end main

   void utfoer (int antT) {
      Thread [] t = new Thread [antT];
      for (int i =0; i< antT; i++)
         ( t[i] = new Thread(new Arbeider(i))).start();
     try {
        for (int i =0; i< antT; i++) t[i].join();
     } catch(Exception e) {}
   } // end utfoer

   class Arbeider implements Runnable {
      // lokale data og metoder B
      int ind;

      void sync() {
         try{
            b.await();
         } catch (Exception e) { return;}
      }

      public Arbeider (int in) {ind = in;};

      public void run() {
         sync();
         System.out.println("A"+ind);
         sync();
         System.out.println("B"+ind);
      } // end run
   } // end indre klasse Arbeider
} // end class Problem
```